COMMUNICATION

7(18), November 1, 2013

ISSN
2278–5469
EISSN
2278–5450

Discovery

Java database connectivity

Lalit Kumar Poddar☼, Tushar Arora, Inderjeet Singh, Tarun Kumar

Dept. of Computer Science, Dronacharya College of Engineering, Farukh Nagar, Gurgaon, Haryana

☼**Correspondence:**
Dept. of Computer Science, Dronacharya College of Engineering, Farukh Nagar, Gurgaon, Haryana; E-mail: lalit.dar4@gmail.com

Publication History
Received: 11 September 2013
Accepted: 17 October 2013
Published: 1 November 2013

Citation
Lalit Kumar Poddar, Tushar Arora, Inderjeet Singh, Tarun Kumar. Java database connectivity. *Discovery*, 2013, 7(18), 8-12

ABSTRACT

JDBC, often known as Java Database Connectivity, provides a Java API for updating and querying relational databases using Structured Query Language (SQL). Java Database Connectivity is a standard SQL database access interface, providing uniform access to a wide range of relational databases. It also provides a common base on which higher level tools and interfaces can be built. This comes with an "ODBC Bridge". The Bridge is a library which implements JDBC in terms of the ODBC standard C API. This paper is designed for Java programmers with a need to understand the JDBC framework in detail along with its architecture and actual usage. This paper will bring you at intermediate level of expertise from where you can take yourself at higher level of expertise.A serious problem facing many organizations today is the need to use information from multiple data sources that have been developed separately. Conflicts in the structure and semantics of these disparate data sources create major obstacles to effective use.

Keywords: JDBC, Java, Database

1. INTRODUCTION

Java started as an elegant and promising Web programming language. Thereafter, its transition to hard-core computing environment is a phenomenal feat. Apart from its significance on client-side programming, Java has been outstanding in developing mission-critical enterprise-scale applications and hence its immense contribution to server-side computing gets all round attention. Thus Java as a programming language for enterprise computing has been doing well for the past couple of years. As every enterprise includes a database, Sun Microsystems has to come with necessary features for making Java to shine in database programming as well. In this overview, we discuss about the role of Java Database Connectivity in accomplishing database programming with ease (www.peterindia.net/).

2. NEED/REQUIREMENT

The JDBC™ API was designed to keep simple things simple. This means that the JDBC makes everyday database tasks easy. This trail walks you through examples of using JDBC to execute common SQL statements, and perform other objectives common to database applications. Java applications support many company's Customer and Sales Compensation areas. This is a very unique and complex part of a company's business model. The application requires a large database to store business-related data. The database consists of dozens of tables and millions of rows. It contains transactional data and summarized data. The size of the database grows daily. Database access is an important part of Java application.The application needs to do lots of complex data inquiry and data processing (docs.oracle.com/). When a Java project begins, the company has to decide how to perform database access from within the Java application. Initially, following factors were to be considered.

1) Companies had developers with SQL expertise gained from their work on legacy systems
2) They wanted to use core Java technologies
3) They wanted to take advantage of any industry standards on database access
4) They wanted direct control of SQL statements for performance tuning purposes.

So they decided that JDBC (Java Database Connectivity) was the best fit, for them, to do data access from their application. They knew of other approaches to database access (object mapping, frameworks, other API's, etc.). The other database access approaches were either too new or did not give them the skills they wanted their developers to have (weblogs.java.net).

3. APPROACH TO JDBC

The JDBC API is a Java API that can access any kind of tabular data, especially data stored in a Relational Database.

JDBC helps you to write Java applications that manage these three programming activities:

1) Connect to a data source, like a database
2) Send queries and update statements to the database
3) Retrieve and process the results received from the database in answer to your query

The following simple code fragment gives a simple example of these three steps:

```
public void connectToAndQueryDatabase(String username, String password) {
   Connection con = DriverManager.getConnection(
                    "jdbc:myDriver:myDatabase",
                    username,
                    password);

   Statement stmt = con.createStatement();

 ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table1");

   while (rs.next()) {
      int x = rs.getInt("a");
      String s = rs.getString("b");
      float f = rs.getFloat("c");
   }
}
```

(http://odbms.org)

3.1. Opening a connection to the DB

„ There are two parts to this: „

- **Loading a driver** – we need a driver to allow our Java program to talk to the DB

The first step in using JDBC is to load a driver.

Here are some examples:

- The IBM DB2 driver:

```
Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");
```

- The SUN JDBC/ODBC Bridge driver:

```
Class.forName("sun.jdbc.JdbcOdbcDriver");
```

There are 4 categories of driver

3.1.1 Type 1 JDBC-ODBC Bridge (Native Code)

- provides a Java bridge to ODBC
- implemented in native code and requires some non-Java software on the client.

„ 3.1.2 Type 2 Native-API (Partly Java)
- uses native code to access the DB with a thin Java wrapper
- can crash the JVM
„ 3.1.3 Type 3 Net-protocol (All Java)
- defines a generic network protocol that interfaces with some
- middleware that accesses the DB
„ 3.1.4 Type 4 Native-protocol (All Java)
- written entirely in Java

- „ **Opening the connection itself**

„ Next, the driver must connect to the DBMS:
„ The object "con" gives us an open database connection

- Connection con = DriverManager.getConnection("jdbc:db2:TEST", "db2admin", " db2admin ");
(http://odbms.org)

3.2. Creating JDBC Statements

A Statement object is used to send SQL statements to the DB. First we get a Statement object from our DB connection con.

Statement stmt = con.createStatement( );
(http://odbms.org)

3.3. executeUpdate (String sql)

Use the executeUpdate() method of the Statement object to execute DDL and SQL commands that update a table (INSERT, UPDATE, DELETE).

```
String createProductTable = "CREATE TABLE PRODUCT " +
 "(NAME VARCHAR(64), " +
"ID VARCHAR(32) NOT NULL, " +
"PRICE FLOAT, " +
 "DESC VARCHAR(256), " +
"PRIMARY KEY(ID))";
stmt.executeUpdate( createProductTable );
```

Be careful to always put spaces in the SQL string at the right places.

Example: inserting rows

```
import java.sql.*;
class InsertProducts
{
public static void main(java.lang.String[ ] args)
{
try
{
Class.forName( "COM.ibm.db2.jdbc.app.DB2Driver" );
String url = "jdbc:db2:TEST";
Connection con = DriverManager.getConnection( url, "db2admin", " db2admin " );
Statement stmt = con.createStatement();
stmt.executeUpdate("INSERT INTO PRODUCT " + "VALUES ( 'UML User Guide', " + "'0-201-57168-4', 47.99, 'The UML user guide')" );
stmt.executeUpdate("INSERT INTO PRODUCT " + "VALUES ( 'Java Enterprise in a Nutshell', " + "'1-56592-483-5', 29.95, 'A good introduction to J2EE')" );
con.close();
stmt.close();
}
catch( Exception e )
{
e.printStackTrace(); }
}
}
```
(http://odbms.org)

3.4. Prepared statements

„ If we want to execute the same SQL statement several times, we can create a Prepared Statement object

- At the point of creation of a Prepared Statement object the SQL code is sent to the DB and compiled. Subsequent executions may therefore be more efficient than normal statements.

- Prepared Statements can take parameters.

Example :

```
import java.sql.*;
class PreparedStatementTest
{
public static void main(java.lang.String[ ] args)
{
try
{
Class.forName( "COM.ibm.db2.jdbc.app.DB2Driver" );
Connection con = DriverManager.getConnection( "jdbc:db2:TEST", "db2admin", " db2admin " );
PreparedStatement findBooks = con.prepareStatement( "SELECT NAME FROM PRODUCT WHERE NAME LIKE ? " );
findBooks.setString( 1, "%Java%" );
ResultSetrs = findBooks.executeQuery();
while ( rs.next() )
{ System.out.println("Name: "+ rs.getString( "NAME" ) ); }
```

```
findBooks.setString( 1, "%UML%" );
rs = findBooks.executeQuery();
while ( rs.next() )
{ System.out.println("Name: "+ rs.getString( "NAME" ) ); }
findBooks.close();
con.close();
} catch( Exception e )
{ e.printStackTrace(); }
}
```

(http://odbms.org)

3.5. Result Set

The SQL statements that read data from a database query return the data in a result set. The SELECT statement is the standard way to select rows from a database and view them in a result set. The*java.sql.ResultSet* interface represents the result set of a database query. A ResultSet object maintains a cursor that points to the current row in the result set. The term "result set" refers to the row and column data contained in a ResultSet object.

```
String queryLehigh = "select * from Lehigh";
ResultSet rs = Stmt.executeQuery(queryLehigh);
while (rs.next()) {
        int ssn = rs.getInt("SSN");
        String name = rs.getString("NAME");
        int marks = rs.getInt("MARKS");
```

(http://odbms.org)

3.6. Closing JDBC Connection

At the end of your JDBC program, it is required explicitly close all the connections to the database to end each database session. However, if you forget, Java's garbage collector will close the connection when it cleans up stale objects. Relying on garbage collection, especially in database programming, is very poor programming practice. You should make a habit of always closing the connection with the close() method associated with connection object.To ensure that a connection is closed, you could provide a finally block in your code. A *finally* block always executes, regardless if an exception occurs or not.

To close above opened connection you should call close() method as follows:

```
conn.close();
```

Explicitly closing a connection conserves DBMS resources, which will make your database administrator happy.

(http://odbms.org)

4. ADVANTAGES OF JAVADATABASE CONNECTIVITY

- Can read any database if proper drivers are installed.
- Creates XML structure of data from database automatically.
- No content conversion required.
- Query and Stored procedure supported.
- Can be used for both Synchronous and Asynchronous processing.
- Supports modules (www.scn.sap.com)

5. DISADVANTAGES OF JAVADATABASE CONNECTIVITY

- Correct drivers need to be deployed for each type of database
- Cannot update or insert multiple tables with sequence (Sequence is always random)
- Need to Correctly open and close Database connection (www.scn.sap.com)

6. CONCLUSION

JDBC can be used to connect to relational databases from Java programs. It allows all basic CRUD operations and can handle sophisticated multi-tier, multi-vendor environments. It copies amounts of error handling required. JDBC can unintentionally create a dependency to a specific database vendor. This research explored the role of JDBC in providing solution developers with enterprise-wide database connectivity capabilities and the JDBC architecture. We studied the relationship between JDBC and Java. We learned about various database systems and how JDBC was designed to work with relational DBMSs and the relational command language, SQL. We learned about two application models. Finally, we learned about how JBDC was designed to leverage the power of Java.

Specifically, we learned to:

1) Distinguish the role and place of JDBC among the Java technologies
2) Differentiate between DBMS types
3) Explain the relational model
4) Describe design considerations for JDBC and ODBC in a solution
5) Explain what SQL is and its role in database processing
6) Explain JDBC as it functions in two system designs
7) Describe the capabilities of Java and a DBMS used with JDBC (www.ewebprogrammer.com/)

REFERENCE

1. eWebProgrammer, Java Database Connectivity, Lesson 9, Enterprise Wide Conclusion
2. Java.net, The Source for Java Technology Collaboration, Why JDBC

3. JDBC,Dr. Jim Arlow, Clear View Training Limited, www.clearviewtraining.com
4. SAP Community Network, Murugavel S, Advantages and Disadvantages of Jdbc
5. The Java Tutorials, JDBC Architecture, Two-tier and Three-tier Processing Models, docs.oracle.com
6. What is the difference between JDBC and ODBC?, www.toostep.com